Contracts for Evolving Systems

Christoph Etzien, Tayfun Gezgin, Sibylle Fröschle, Stefan Henkler  
Institute for Information Technology (OFFIS)  
Oldenburg, Germany  
{etzien, gezgin, sibylle.froeschle, henkler}@offis.de

Achim Rettberg  
Carl von Ossietzky University Oldenburg  
Oldenburg, Germany  
{achim.rettberg}@informatik.uni-oldenburg.de

Abstract—In this work we address evolving systems, which are basically collaborative and distributed systems building up a larger scale system of system (SoS). These systems are able to adapt the current architecture to some changes in the environment. Constituent systems of a SoS, which represent the basic elements of our modeling approach, operate with different degrees of freedom and as a result the self-adaptation and cooperation between a set of constituent systems is driven by local needs.

Based on our former work [11], we propose a well-defined modelling approach for SoS capturing both static and dynamic aspects. The aim is to address on the one hand the required flexibility to adapt the systems during run-time, and on the other hand to guarantee that the SoS reacts still in a safe manner. For this, we will use the contract paradigm for both the specification of legal configurations of the SoS, and to specify the dynamicity model, describing how the SoS architecture can change during run-time. Further, we depict how to adapt a system level analysis technique in order to check the dynamicity model against the invariants of the SoS. With this, we are able to determine, whether the SoS can reach some critical configurations. This enables us to modify the dynamicity model in an adequate manner.

Index Terms—Systems of Systems; Evolving Systems; Real-Time Adaptive Systems; Contracts-based Design; Dynamicity

I. INTRODUCTION

Systems of Systems (SoS) have become an important topic in the field of researches in the resent years. Mark Maier defined set of characteristics which help to distinguish SoS from complex systems in [30]: A SoS performs functions which are not achievable by single systems. This criterion is called emergent behavior. The constituent systems of a SoS are geographically distributed, such that all participating systems need to exchange data in a distributed and remote way. The evolutionary development defines that services and purposes change at run-time. Managerial independence specifies that systems are designed independently and are managed for their own purposes. Operational independence states that systems behave active to achieve their goals. The more a system exhibits these five characteristics, the more it is a SoS.

Based on these characteristics, we define a meta-model to create a model of a system of systems in this work. This meta-model is an extension of our previous work introduced in [11]. First, we address static aspects of a SoS. The major building blocks are constituent systems representing elements of a SoS, which are independently designed and may operate on their own. Constituent systems come along with a contract-based interface specification, such that the environment can completely abstract from the internal implementation details of this system and rely on the functional and quality properties with respect to their contracts. In our approach a so called service describes elementary actions of the system like for example a service of a fireman to drive a fire truck or to fight a fire including also basic classical real-time patterns like a watchdog or an asynchronous message queuing pattern. These services are the basic blocks for roles. Constituent systems cooperate in a well-defined manner through the concept of roles specifying some interaction protocols. When cooperation structures between constituent systems are established, these systems have to take on specific role behaviors.

One major aspect of a SoS is its dynamic evolution, i.e. the change of its architecture during run-time. When certain architectural constraints are met and environmental events occur, changes will be initiated by some constituent systems. For this, we introduce a contract-based specification formalism to capture the dynamicity model of a SoS. Activations for changes are captured by the means of weak assumptions. The guarantee part of dynamicity contracts encapsulates changes in the SoS architecture. We define two types of dynamicity contracts: Request contracts capture the dynamic changes where more than a single system is influenced and some agreements have to be realized between a set of systems. Action contracts on the other hand directly affect the SoS architecture. This dynamicity model describes all possible evolutions of the constituent systems and parts of the considered SoS. Such a model could be created based on prognosis on possible future evolutions of the SoS. As an example, statistical data could be used to do a prognosis on the future traffic density in a district of a city.

Besides the dynamicity specification, we will consider static contracts of SoSs defining invariants and constraints on the dynamic evolution such as reconfiguration times. The set of static contracts of a SoS define on the one hand all legal configurations, i.e. configurations adhering to these invariants, and on the other hand a set of legal dynamicity models, which satisfy the constraints on dynamical evolutions. To analyze dynamicity contracts we present an adaption of our system level techniques described in [12].

This work was partly supported by European Commission funding the Large-scale integrating project (IP) proposal under ICT Call 7 (FP7-ICT-2011-7) Designing for Adaptability and evolution in System of systems Engineering (DANSE) (No. 287716),
SoS. In order to capture the assumption and guarantee parts of contracts, we introduce the language of timed first-order linear temporal logic (TFOLTL) in Section III. The usage of contracts for SoS is illustrated in Section IV, while in Section V the analysis of the dynamicity model is defined. Section VI exemplifies our approach on an example of an emergency response scenario. We give a conclusion and an outlook in Section VII.

A. Related Work

Modeling languages of systems of systems has recently been the topic of several works. UPDM [14] is a unified Profile for DoD Af (Department of Defence Architecture Framework) and MODAF (Ministry of Defence Architectural Framework). It supports the capabilities to model architectures of complex systems, system of systems, and service oriented architectures. In UPDM, model elements are organized in various viewpoints or views. Each view or viewpoint focuses on certain criteria of the design, e.g. the service-orientated view is a description of hierarchical services. The framework is very complex and not intuitive. Beside milestones, no dynamicity aspects of systems of systems were considered in this framework.

Ultra large scale systems are the topic of [25], where the main characteristics are captured and specified, e.g. decentralized control, conflicting requirements, and continuous evolution. In [27] some major issues in self-coordinating systems are depicted. The main statement is that a tight integration of all disciplines in the development process of such large scale self-coordinating systems has to be established. An approach for the design and analysis of multi-agent systems was presented in [13]. Agents are able to sense and manipulate specific aspects of the environment. Sets of agents form community types, which interact in the modeled environment with some interaction specifications. In [15] self-adaptive systems were presented. Initially a system architecture with defined components, their interfaces, and a set of coordination pattern is given. Coordination pattern define protocols between components via roles. A role behavior is specified via Real-Time Statecharts, which are semantically based on timed automata. Reconfigurations are defined via graph transformation rules and are initiated by environmental changes.

Automatic verification of the real-time behavior including the reconfiguration is supported by CHARON [2], [20], Masaccio [16], and MECHATRONIC UML (e.g. [7], [6], [18]). There are some approaches for modeling the structural aspects of adaptive systems [28], [23], [17], [26], [22] or the behavioral aspects [31], [1], [21], [8] but none of them consider both aspects [5]. This was first considered by the approach presented in [18].

The work of Bauer et al. [3] focuses the analysis of safety and liveness properties of dynamic systems, which have an unbounded number of processes and data with infinite domains. The use case for this approach are car platoons. Dynamical changes of modes of cars participating in a platoon are modeled by a discrete transition system. Reached configurations are represented by graphs. They define an abstraction technique to reduce the unbounded number of constituent systems and reachable architectures to a finite set of clusters. A similar work for car platoons was worked out from Wachter and Westphal [29], where only a finite number of processes is represented exact and the rest is abstracted into one dedicated summary process.

II. Modelling Systems of Systems

In this section we define a modelling language in order to capture static and dynamic aspects of a system of systems (SoS), which is an extension to our previous work [11]. Contracts will play an important role in our language. They will be used to specify both the behaviour of constituent systems (CS) and evolutionary aspects. Contracts will help to treat the SoS modelling problem in the same manner as the modelling of a system architecture. Our proposed meta-model (see Figure 1) takes into account that CSs have several degrees of freedom which not apply to reactive systems. Maier [30] had identified five such degrees in order to distinguish between complex systems and SoS. In this paper we focus on the operational independence of CSs and the evolutionary development of SoSs itself.

- Operational independence means that the CS are acting independently rather than a central controller, and that those systems are rather active than re-active. The most important aspect of this is that connections like communication with other CSs can be initiated by the CSs themselves and do not need a central control entity.
- Evolutionary development, or short evolution, is the change in the architecture of the SoS. This includes dynamically changing interaction connections between CSs as well as the creation and removal of CSs to/from the SoS.

We address these properties in our meta-model by defining protocols which support the creation and deletion of connections between the CSs and also by specifying the CS behaviour with respect to their type. This allows to derive the SoS

![Fig. 1. SoS meta-model](image-url)
behave from the current architecture and the behavioral specification of the CSs. Especially, the cooperation behavior is explicitly defined by the concept of roles.

Constituent systems define the main building blocks of our approach. They are independently designed and their instances can operate autonomously. In a larger context, the independent development leads to a limited knowledge about the internal details of the CSs. We assume that the internal structure is completely unknown but all relevant interfaces are known. The interfaces of CSs come along with a specification in terms of contracts, which specify functional and quality properties of the offered services. The environment can rely on these properties, such that each CS can be treated as a black-box. We distinguish between to subsets of these contracts. The static contracts are contracts in the classical sense arguing (only) about exchange on the interface of the CS. The dynamic contracts addresses the interaction with other CSs from a CS level point of view. This means that the SoS architecture or a subset of it is taken into account in the assumption part and the activation of changes of the SoS architecture are part of the guarantee. An example from the perspective of a CS $A$ is that if there exists another CS $B$, a request to establish a communication connection between $A$ and $B$ is emitted by $A$. If the response of $B$ is positive, a communication connection is established and thereby the SoS architecture is changed.

We add the concepts of role, service and capability as subsets of specified behavior to capture SoS level behavior. Each service of a CS is a subset of its specified behavior. A role is associated to a port which is connected and defines which services are performed on this connection end. A capability defines the set of potential services the CS is able to perform, when a cooperation with another CS is started and a connection is established. As an example for roles, consider Figure 2 where the two roles fire fighter and water transporter are illustrated and specified in terms of timed automata.

The different kinds of contracts required to capture not only system behavior but also adaptiveness in terms of reconfiguration is detailed in Section IV. The next section deals with the temporal logic which is used to define the semantics of the contract language.

III. TIMED FIRST-ORDER LTL

In this section we define a real-time extension for first-order linear temporal logic (TFOLTL), which is based on the formalism introduced in [10].

A signature defines the relevant objects and relations of the considered problem domain, e.g. a set of natural numbers and their relations between those numbers. Formally, let $SIG = (S, F, P)$ be a signature with a set of types (or sorts) $S$, a set of function symbols $F$, and a set of predicate symbols $P$, both typed over $S$. For each $s \in S$ we have a finite set $X_s$ of variables. Let $L_{FOL}(SIG)$ denote the classical language of first-order logic over $SIG$.

A structure $S$ for a signature $SIG = (S, F, P)$ determines the domains for each sort $s \in S$ by $|S|_s$, and determines interpretations for the function and predicate symbols in the classical way. A variable valuation $\xi$ maps each variable $x \in X_s$ for all $s \in S$ to an element of $|S|_s$, in short $\xi(x) \in |S|_s$.

A signature $SIG$ is extended to a temporal signature, which is given by $TSIG = (SIG, R, A)$, where $R$ is a set of so called variable function, and $A$ is a set of flexible predicates, both again typed over $S$. The elements $R$ and $A$ are called flexible, as their evaluation is state dependent. The language $L_{FOLTL}(TSIG)$ of linear temporal first-order logic over $TSIG$ will be defined in the following.

Terms and atomic formula of $L_{TFOLTL}(TSIG)$ are the same as in classical predicate logic, extended by terms and formula over the flexible elements. Note, that especially each $r \in R$ is a term and each $f \in A$ is an atomic formula. The formula of $L_{TFOLTL}(TSIG)$ are then inductively defined:

- Each atomic formula is a formula.
- If $f_1, f_2$ are formula, then $f_1 \Rightarrow f_2, \Box f_1, \Box [t_1, t_2] f_1, \bigotimes f_1$, and $\dot{f}_1[1_{t_1}, t_2] f_1$ are formula.
- If $f$ is a formula and $x \in X$, then $\exists x f$ is a formula.

In the following we will give the semantics of $L_{TFOLTL}(TSIG)$.

Let $\tau : N \rightarrow \mathbb{R}$ be a monotonically increasing time sequence, i.e. for all $i \geq 0 : \tau_i \leq \tau_{i+1}$. Let $\Sigma$ be an alphabet. An infinite sequence $\sigma = \sigma_0 \sigma_1 \ldots$ of symbols from $\Sigma$ is called a trace. A timed trace over an alphabet $\Sigma$ is a infinite sequence $(\sigma, \tau) = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\ldots$, where $\sigma$ is a trace and $\tau$ is a time sequence. In our context, such timed traces will define timed sequences of valuations of flexible elements.

Let $\eta : R \cup A \rightarrow |S| \cup \{tt, ff\}$ be a valuation function for the flexible elements, where $\eta(r) \in |S|_r$ for each $r \in R_s$ (i.e. flexible predicate of sort $s$) and $\eta(a) \in \{tt, ff\}$ for each $a \in A$. The set of all valuations for $R \cup A$ is given by $E$. Let $K = (S, W)$ be a temporal structure for a temporal signature $TSIG = (SIG, R, A)$, where $S$ is a structure for $SIG$, and $W$ a trace over $E$. 
Let \( \xi \) be a variable valuation function, \( TSIG = (SIG, R, A) \) a temporal signature, and \( K = (S, W) \) a temporal structure. For a term \( t \) and an atomic formula \( f \) in \( LTFOLTL(TSIG) \) the definitions for \( S^{(\xi, a)}(x) \) and \( S^{(\xi, a)}(r) \) are given for each \( a \) are:

- \( S^{(\xi, a)}(x) = \xi(x) \) for \( x \in X \),
- \( S^{(\xi, r)}(r) = \eta_{r}(r) \) for \( r \in R \),
- \( S^{(\xi, a)}(a) = \eta_{a}(a) \) for \( a \in A \).

For a formula \( F \) in \( LTFOLTL(TSIG) \) and each \( i \in \mathbb{N} \)
\( K^{\xi}_{i}(F) \) is inductively defined:

- \( K^{\xi}_{0}(A) = S^{(\xi, a)}(A) \) for each atomic formula \( A \).
- \( K^{\xi}_{i}(A \Rightarrow B) = tt \) iff \( K^{\xi}_{i}(A) = ff \) or \( K^{\xi}_{i}(B) = tt \).
- \( K^{\xi}_{i}(\square A) = tt \) iff \( K^{\xi}_{j}(A) = tt \) for each \( j \geq i \).
- \( K^{\xi}_{i}(\bigcirc A) = tt \) iff there is such \( j \geq i \) where \( \tau_{1} + t_{1} \leq \tau_{j} \leq \tau_{1} + t_{2} \) and \( K^{\xi}_{j}(A) = tt \).
- \( K^{\xi}_{i}(\forall_{[t_{1}, t_{2}]} A) = tt \) iff there is such \( j \geq i \) where \( \tau_{1} + t_{1} \leq \tau_{j} \leq \tau_{1} + t_{2} \) and \( K^{\xi}_{j}(A) = tt \).
- \( K^{\xi}_{i}(\exists_{x}.A) = tt \) iff there is a variable valuation \( \xi' \) with \( \xi'_{\sim x} \equiv \xi \) and \( K^{\xi'}_{i}(A) = tt \).

Note, that \( \xi'_{\sim x} \) is defined as \( \xi'(x) = \xi(x) \) for all \( X \setminus \{x\} \). If for all \( i \in \mathbb{N} \) and all variable valuations \( \xi \) it holds that \( K^{\xi}_{i}(A) = tt \), then \( K(A) = tt \).

IV. CONTRACTS FOR SOs

In this section, we will detail the question of how to use contracts in the context of SOs. The idea of contracts is inspired by Bertrand Meyer’s programming language Eiffel and its design by contract paradigm [24]. Generally, a contract is a pair consisting of an (strong) assumption (A) and a guarantee (G). A strong assumption specifies the context of a system, i.e. the environment from the point of view of the system should behave. Only if the assumption is fulfilled, the system will behave as guaranteed.

In [4] we extended contracts by the means of weak assumptions, which partition the allowed context of the system specified by the strong assumption. The intuition is, that if the integration context is able to ensure a weak assumption of a component, this component may guarantee a more restricted behavior.

We will adapt the contract-based design approach, such that evolutions of an architecture can be rigorously specified. For this, we will use strong assumption-guarantee-pairs in order to characterize invariants on both system and SOs level, i.e. characteristics which always have to hold. Weak assumptions-guarantees-pairs will be used in order to characterize the dynamicity model of a given SOs, which describes how the SOs can evolve.

A. Static Specification

On system level, the interface specification can be rigorously defined by contracts. Thereby, the allowed design contexts of a system is captured by strong assumptions: A violation of these assumptions will lead to an out-of-specification usage of the corresponding system. For allowed design contexts, the guarantee part of the contract allows to characterize both functional and quality properties (e.g. safety and real-time) of the offered services. A system can be decomposed into several components (or parts), again equipped with their interface specification and implementation. By specifying interface properties of systems with contracts we have several proof obligations:

- The satisfaction check deals with the question, whether the implementation of a system adheres to its specification.
- Compatibility: A crucial question in the design of systems is whether the parts are working correctly together, i.e. the composition is correct.
- Virtual Integration: This verification task refers to the question whether a design component can be virtually integrated into the surroundings of an other component.

For more details about proof obligations of contract based design on system level please refer to [19].

Contracts on system level refer to observables, which can be measured. With a contract based interface specification one can abstract from the internal details of a system. The environment can rely on the functional and quality guarantees specified by the corresponding contracts.

Contracts on SOs level can help in order to specify invariant constraints. Contracts on this level can specify, that if changes in the SOs architecture occur, certain functional and quality aspects still do hold. As an example, when a fire truck of a fire department fails, the SOs should reconfigure itself in such a way, that the guaranteed service in timely handling a fire is not affected.

A difference between system level specifications and SOs level invariants is that besides observables, a contract can refer to architectural aspects (introduced in Section II). We can specify that certain architectural configurations should not be reached through reconfiguration. As an example, a communication between two constituent systems using incompatible roles should never occur. As ports and relations are annotated by roles, we are able to check such properties.

Invariants on SOs level specify the allowed reachable set of configurations of the SOs, including the allowed set of architectures and constraining their functional and quality aspects. In the next subsection, we will specify the dynamic evolution of a SOs, i.e. how the SOs architecture can change.

B. Dynamicity Specification

Besides the static part, one major aspect of systems of systems is its dynamic evolution, i.e. the change of its architecture during run-time. For this, we need a model describing the evolution of the constituent systems and parts of the considered
SoS. This model is created based on the prognosis of possible future evolutions of the SoS. As an example, statistical data could be used to do a prognosis on the future traffic density in a district of a city.

We will use contracts in order to specify dynamical changes on the architecture. The intuition of a dynamicity specification is not to say, that under some assumptions on the environment a guaranteed behaviour can be offered. The assumption shall rather specify an activation condition to initiate a change. This exactly corresponds to a weak assumption rather than a strong assumption.

A dynamicity contract $c_{dyn}$ consists of a weak assumption $a_w$ and a guarantee $g$. It describes an assumption on the SoS architecture, e.g. in terms of architectural pattern, and specifies architectural changes in the guarantee part. Thus, in contrast to static contracts, the guarantee part of a dynamicity contract implies changes in the architecture. The semantics of a dynamicity contract $c_{dyn}$ is the implication relation $a_w \Rightarrow g$ where $a_w$ is the weak assumption and $g$ the guarantee.

In order to actually change the SoS architecture, constituent systems have to be able to take a certain role in an interaction with other constituent systems. These roles specify protocol behaviors in terms of requested and provided services between participating systems. As an example, if the capacity of a fire department is exceeded, a request for cooperation to its neighbors is send. Depending on the reaction of the neighbors, new cooperation relations are established. For this, we introduce two classes of dynamicity contracts, i.e. request contracts and action contracts. The former are those contracts, where more than a single constituent system is involved in a change, i.e. contracts which initiate the creation of a new connection between a set of constituent systems. The second type of dynamicity contracts directly affects the SoS architecture (e.g. the establishment of a new connection).

To specify contracts by TFOLTL, we set the flexible predicates to $A = A_{role} \cup A_{action} \cup A_{env}$, where $A_{role}$ defines the set of role predicates, $A_{action}$ a set of possible action predicates, and $A_{env}$ a set of environmental action predicates. The set of possible actions $A_{action}$ describes the possible changes on the architecture, e.g. adding new edges or constituent systems. Environmental actions $A_{env}$ describe characteristics of the environment with respect to the constituent system, e.g. a change in the service load. With role predicates $A_{role}$ requests for cooperation are enabled. Note, that specific predicates depend on the context of the SoS.

With the set of flexible predicates, we are able to formulate our dynamicity contracts: The guarantee part of action contracts is defined over $A_{action}$. Consider for example, that a fire department needs new fire trucks $FT$ and has the corresponding budget. Then this constituent system would be annotated by the action contract $a_w : budgetAvailable \land need(FT)$, $g : add(FT_i)$, where $add(FT_i) \in A_{action}$ and $need(FT)$, $budgetAvailable \in A_{env}$. The guarantee part of request contracts is defined over $A_{role}$. When a cooperation partner is needed, a request in the guarantee part of the corresponding contract is performed. When there is a constituent system, which can take on this requested role, it can establish this co-operation. As an example, consider a fire department, which sends a request to all neighboring departments for cooperation for a special role. The fire department would have a contract with the guarantee $g : \forall FD \in S,neighbors(FD_i),req(roleX,FD)$, where $FD_j \in S,neighbors(FD_i) \in F,req(roleX,FD) \in A_{role}$ for all $FD \in S$. If there exists a constituent system annotated with a contract containing the weak assumption part $a_w : \exists FD. req(roleX,FD)$ then this system can establish a new connection for co-operation. In Section VI we will give a detailed example for this kind of contracts.

The set of dynamicity contracts of a constituent system defines its possible evolution during run-time. The set of all dynamicity contracts of all constituent systems is the evolution description of the SoS.

V. PROGNOSIS ON CRITICAL CONFIGURATIONS

The dynamicity model described in the previous section specifies all possible evolutions of the considered SoS. When static contracts are defined, we have to check, whether the set of reachable configurations adhere to these static contracts. A configuration is called critical configuration, when it does not adhere to the static contracts of the SoS.

Static contracts mainly consists of two different types. The first type is invariant contracts. Each configuration has to adhere to the set of invariant contracts, i.e. for each reached configuration a check against these contracts is performed. Invariants are used to describe certain quality aspects and architectural configurations. When several CSs are working together to realize a higher level service, some quality guarantees could be required by invariants. For example, to timely fight a fire, several services from different constituent systems have to be performed. This time bound has to be guaranteed irrespective of the current available resources and cooperation relations between them.

The second kind of specification addresses the dynamicity model: Given a dynamicity model $D$ and a specification $I$ constraining dynamical aspects, the verification question is, whether $D$ adheres to $I$. Constraints on dynamic aspects are verified based on the set of sequences of configurations specified by $D$. More specifically, the guarantee part of these contracts are formula over $A_{action}$ rather than $P$ (predicate symbols) or $A_{role}$. These contracts specify that certain changes in the SoS architecture have always to be finished within a certain amount of time. An example for this kind of contracts is, that certain reconfiguration times are demanded.

In the following, focus on the analysis of real-time constraints for the dynamicity specification of a SoS. Such timing constraints capture allowed reconfiguration times of the SoS. To analyse constraints on dynamicity specifications, we will use our system level analysis technique introduced in [12]. For this, we will treat the dynamicity specification as a behavioral model of the SoS describing the possible evolution.

The approach in [12] combines both analytical and model checking methods for end-to-end deadlines. For this, the state
space of the entire system architecture is constructed in a compositional manner. This state space is determined by the allocated scheduling policies, the input behaviors which trigger the allocated tasks on the resources, and the execution behaviors of the tasks itself. The state space of a resource is represented by a so called symbolic transition system, which explicitly encapsulates all reachable states of this resource. Based on this state space, response times for all tasks are determined, and safety properties can be verified. Some assumptions are done in order to realize such an end-to-end analysis. First, the input behaviour of independent tasks or rather services (in our case) are triggered periodically. Further, a priority based protocol with preemption is considered, which orders the task execution on a resource. Dependent tasks are directly activated by termination times from tasks on which they depend.

In order to use this analysis technique, we consider that constituent systems represent resources itself or posses a set of resources - e.g. the constituent system fire department contains a set of fire brigade resources - and consider their services as tasks. Further, we assume that a resource can only execute services in a sequential manner, i.e. a resource cannot handle a set of services in parallel. Services are prioritized and may preempt other services.

With this set of assumptions, our approach is to transform the dynamic model of a SoS specified by TFOLTL to a network of timed automata. The dynamics of a SoS consists of several dynamicity contracts annotated to constituent systems. For each dynamicity contract of a constituent system we build timed automata, i.e. for each constituent system we get a set of timed automata which describe its dynamic behaviour. From such a timed automaton network, we build the symbolic transition system in a compositional manner as described in [12], and determine the reachable states and check the violations of timing constraints as specified in the SoS dynamics constraints. Note, that this is ongoing work. Currently, we are transforming a set of contracts specified in TFOLTL to timed automata and establish a compositional reachability check.

VI. CASE STUDY

Derived from the Concept Alignment Example used in the DANSE project\(^1\), we take the fire fighting service as our case study illustrated in Figure 3. Each fire department hosts several fire brigades and manages the incidents in cooperation with other fire departments and the crisis control centers in a city or region. To simplify the example we focus on one city with only one crisis control center. The city is partitioned into several districts. Each district is managed by one fire department. Incident calls are handled by the control center and forwarded to the appropriate fire department.

For the fire fighting service a global timing and quality of service requirement is defined by the government which restrict the time between the incident call and the arrival of rescue units. This requirement arises the necessity for cooperation between fire departments, when a single department is not capable in guaranteeing the demanded quality.

In our example the fire departments may need to cooperate in order to handle an incident. If e.g. more than one incident occurs at the same time in the same district, the responsible fire department may need to be supported by another fire department. If this cooperation is established, the SoS architecture is changed. The initialization of such an cooperation is triggered by the fire department which needs support from another. In this case the fire department is in the role of a manager for a certain incident and tries to change the behavior of another (e.g. neighbor) fire department to perform the supporter role. To initiate such connection a communication protocol is defined using request contracts and the establishment of this connection is realized by action contracts.

In the following, we will formulate the dynamic contracts using TFOLTL. In our example we have the sorts \( S = \{ FD, Port, Capacity, Size, Location, Role \} \). The variable we need to describe the example are:

\[
\begin{align*}
X_{FD} &= \{ fd_i | i \in \{0, \ldots, n\} \}, \\
X_{Capacity} &= \{ c \}, X_{Size} = \{ s \} \text{ over domain } N \\
X_{Location} &= \{ l \} \text{ over domain } N \times N \\
X_{Role} &= \{ supporter, manager \} 
\end{align*}
\]

The sort \( FD \) represents a fire department with interfaces of type \( Port \). The set \( X_{FD} \) models the set of fire departments in our SoS. Further, we need a sort called \( Role \) defining the set of cooperation protocols. In this example, we have two roles called supporter and manager for handling incidents in a cooperated fashion. The sorts \( Capacity \) and \( Size \) will be used to define available capacities of fire brigades and the size of an incident. The sort \( Location \) enables to describe places of incidents. Next, we define flexible predicates:

\[
\begin{align*}
A_{env} &= incident( Size, Location) \\
A_{role} &= req( Role, FD, FD, Size) \\
A_{action} &= edge( FD, Role, Port, FD, Role, Port) 
\end{align*}
\]

The environmental action predicate \( incident( Size, Location) \) states, whether at a certain location an incident with a certain size is present or not. The role predicate

---

\(^1\)www.danse-ip.eu
reg\((\text{Role, FD, Port, FD, Port, Size})\) enables to specify request contracts. With this, a FD can request another FD to cooperate and to take on a specific role at a certain port. The parameter of type \(\text{Size}\) specifies a quantity for the requested resources (fire brigades). Further, we have a action predcate \(\text{edge}(\text{FD, Role, Port, FD, Role, Port})\) enabling to specify an edge between the interfaces of two FDs adhering to specific roles are be created. Besides flexible predicates we define the set flexible functions \(R = \{\text{resp, utilization, coops, neighbor}\}\) with:

\[
\begin{align*}
\text{resp:} & \quad \text{Location} \to \text{FD} \\
\text{cap:} & \quad \text{FD} \to \text{Capacity} \\
\text{coops:} & \quad (\text{FD} \times \text{Location}) \to \{\text{FD}\} \\
\text{neighbor:} & \quad \text{FD} \to \{\text{FD}\}
\end{align*}
\]

The function \(\text{resp}\) specifies which fire department is responsible for which location, \(\text{cap}\) specifies the available capacity of a fire department, \(\text{neighbor}\) defines for each fire department the set of fire departments which are adjacent, and \(\text{coops}\) specifies for each fire department \(fd_i\) and each location \(l\) the set of fire departments, which are supporting \(fd_i\) at location \(l\) (if there is an incident). We define the dynamicity contracts for a fire department \(fd_i\):

\[
a_w: \exists s. \text{incident}(s, l) \land \text{resp}(l) = fd_i \land \text{cap}(fd_i) < s \\
g_t: \forall fd_j \in \text{neighbor}(fd_i). \text{req}(\text{supporter, fd}_j, fd_i, \text{diff}(s, \text{cap}(fd_i)))
\]

This contract specifies, that an incident with size \(s\) at location \(l\) occurs, the fire department is responsible for this location, and its current capacity is exceeded by the needed size to handle this incident, then a request for cooperation is send to all neighbors. That is, a fire department is searched, which can take on the role \(\text{supporter}\) and is able to handle the size \(\text{diff}(s, \text{cap}(fd_i))\), i.e. the residual capacity. To respond to such a request, we need the following contract:

\[
a_w: \exists fd_j. \text{req}(\text{support, fd}_j, fd_i, s) \land s < \text{cap}(fd_i) \\
g_t: \diamond_{[0,1]} \text{edge}(fd_i, \text{supporter, fd}_j, \text{manager})
\]

This contract is active, whenever a request for cooperation from a fire department is received and the needed capacity can be offered. If the assumption holds, an cooperation is established by creating an edge to the requester within some time interval \([0, 1]\). The last dynamicity contract specifies, that if the handling of an incident is completed, then all cooperations should be terminated:

\[
a_w: \neg \text{incident}(s, l) \land \text{coops}(fd_i, l) \neq \emptyset \\
g_t: \diamond_{[0,1]} \forall fd_j \in \text{coops}(fd_i, l). \neg \text{edge}(fd_j, \text{supporter, fd}_i, \text{manager})
\]

With this set of dynamicity contracts, future behaviour of the SoS can be analyzed. Besides this dynamicity specification, we further define constraints with the aid of a static contract:

\[
a_w: \square(\exists fd_j \in \text{neighbor}(fd_i). \text{cap}(fd_i) > \text{minCapacity}) \\
g_t: \exists s. \text{incident}(s, l) \land \text{resp}(l) = fd_i \land \text{cap}(fd_i) < s \\
\Rightarrow \exists t_j \in [t_1, t_2]. \text{edge}(fd_i, \text{supporter, fd}_j, \text{manager})
\]

where \(\text{minCapacity}\) is a constant. This contract specifies, that whenever an incident occurs and the responsible fire department is not able to handle it due to capacity shortcomings, a cooperation should be established within the time interval \([t_1, t_2]\). In order to be able to guarantee such behaviour, we have to demand, that there is always a fire department in the environment, which has sufficient capacity available.

**VII. CONCLUSION AND FUTURE WORK**

In this work we presented a modeling approach for systems of systems, which is an extension to our previous work [11]. We defined statical aspects as constituent systems including their contract-based interface specifications, offered services, and roles. Further, we worked out a specification formalism which allows to capture evolutionary aspects of SoS by so called dynamicity contracts. When certain architectural constraints are met and environmental events occur, changes will be initiated by some CSs. In order to capture the assumption and guarantee parts of contracts, we introduced the language of TFO-LTL. To analyze dynamicity contracts we presented an adaption of our system level techniques worked out in [12].

Currently, we are working on a transformation and implementation of a real-time fragment of TFO-LTL to timed automata in order to reuse our system level analysis techniques to realize both checks of invariant for reachable configurations of the SoS, and to check whether the dynamicity specification adheres to its requirements. An important topic here will be the automatic adaption of the SoS. For this, the dynamicity model will be automatically enriched by further constraints.

In future work, we will define a pattern-based specification language, such that the SoS invariants and dynamicity specifications can be captured in a more user friendly and intuitive manner. The semantics of this pattern language will be defined by the TFO-LTL language. The pattern language will be based on the requirement specification language worked out in [9].

Another direction will be the application of abstraction techniques for our approach, as for example the work of [29] where only a finite number of processes is represented exact and the rest is abstracted into one dedicated summary process. With this, we will investigate the possibility to include local views to our approach.

**REFERENCES**


